# HOW CAN OPERATIONS GET THE APPLICATIONS SOFTWARE THAT THEY WANT ?

R. Bailey, CERN, CH-1211 Geneva 23, Switzerland

*In SL division at CERN, any major development of application software to be used in the control room entails interaction and a division of responsibilities between people from operations, controls, accelerator physics and equipment groups. Providing the suite of applications for the operators to use for efficient exploitation of the machines is no exception to this. Indeed since routine operation accounts for the largest fraction of machine time, it should be regarded as the most important activity. We have found over 10 years that teams led by operations people, making pragmatic use of formal development methods, prove to be very effective in producing the software required for running the machines. This paper examines the* modus operandi *of some operations-driven projects, and contrasts this with some less successful ventures.*

## 1 INTRODUCTION

The successful production of satisfactory application software for driving accelerators has remained a difficult goal for years. This is true over a wide range of laboratories and over the different categories of user. Why is this so ? In an attempt to throw some light on this perennial problem, the question is broken down into three;

- who should be considered the main user of a control system ?
- what features are required ?
- how can the desired functionality be achieved ?

## 2 WHO SHOULD DEFINE A CONTROL SYSTEM ?

Any accelerator control system is used at different times by different personnel. During the latter stages of the construction phase, and later on for detailed investigations of faults, many equipment specialists need to run extensive tests of their hardware. During commissioning, and later on in machine development, accelerator physicists need a wide variety of utilities to measure and manipulate the beam. Once the machine is up and running, operations need a comprehensive suite of applications software for the long-term exploitation of the machine. These different groups of people each have very different and often conflicting requirements, and while a good control system should of course try to satisfy all kinds of user this is rarely, if ever, the case.

Where should the priority be ? Software for equipment testing is needed before software for beam commissioning, which in turn is needed before software for routine operations, and this chronology of course determines the order in which applications have to become available. However, while the priority in terms of the calendar is clear, the priority in terms of machine time is not. The equipment testing and beam commissioning phases are hopefully rather short, amounting to something like six months each. Operational running of the machine, on the other hand, is hopefully rather long, with ten years a typical minimum. Within the years of routine operation there are of course equipment faults to be diagnosed and machine development periods to be exploited, but again these should constitute a small fraction of the whole. In a typical year of LEP operation, for example, the machine time is divided into 70% on routine operation, 20% on machine development and 10% on fixing problems.

Seen in this light (and there are others [1] ), the requirements of the operators should be of prime importance. In fact, routine operation is the reason the machine was constructed in the first place, with commissioning a necessary step along the way. Unfortunately, when planning what will be needed to run a machine, most application development projects concentrate on the earlier but shorter phases, fading out once the machine is successfully commissioned and neglecting completely the longer term. A new initiative, taking considerable time and requiring

new resources, is then needed before operations are able to run the machine effectively and efficiently over many years.

## 3 WHAT FEATURES DO OPERATIONS WANT ? - THE BASICS

Leaving aside for the moment the crucial question of the detailed functionality required to run the machine, two things set operations personnel apart from other users of the control system. Firstly, the scope of their work necessarily covers the whole machine rather than specialist areas, and in this they prefer a high level of standardisation across different applications (and even, in some cases, across different accelerators). Secondly, machine operation happens 24 hours a day, seven days a week, typically for 30 to 40 weeks each year, and as such the 'comfort' provided by the control system is extremely important.

### *A. Standardisation*

Having a suite of applications that conform to well defined standards can contribute enormously to an operator's efficiency. The basic interaction with different applications should be required to follow a set of rules specified by operations. This would enable all the essential functions (start, stop, pop, push, reduce, enlarge, ...) and some slightly deeper functionality (display handling, parameter input, task and sub-task selection, ...) always to be performed in the same way. With this, navigation around the system and a large part of the operator's interaction with it would be standard. One might say that this is obvious, but a lot of software presently used to control SPS, for example, would fail to meet these basic requirements. Why is this so ? The reason for this shortcoming is nearly always due to the fact that the software in question was produced either by, or following specifications from, equipment specialists, and was then made available to operations. Such software may be fine for the individual needs of the equipment specialist, but it does not fulfill the more global requirements for operations.

### *B. Ease of use of interactive applications*

Listed here are but a few of the characteristics of user-friendly applications software:

- Reliability
- Stability to change
- Error reporting
- On-line help
- Execution speed
- Data on a need to know basis

Again one might think that these requirements are obvious, but at three in the morning they often seem to take on an extra significance.

### *C. Fixed displays*

Operations is not all interactive. During stable conditions an accelerator often runs without any operator intervention. What the operator needs at these times is a good fixed display system, giving him at-a-glance access to a rather large amount of data which tell him either that all is running well or that something has gone wrong. Display of information of this kind on a permanent basis, with a regular update, contributes greatly to the operator's comfort.

Much use is made of these facilities in the control room from where both SPS and LEP are controlled. For stable LEP operation there are about 15 fixed display screens, many of them divided into 4 or sometimes more windows, and many of them configurable to suit the operator's needs. For the SPS the number is a little smaller but the system is still regarded as essential for operation of the machine.

### *D. Alarms*

A particular example of a fixed display is the alarm screen, which is again of great importance for both SPS and LEP operation. There are two alarm displays for each machine, and they are used in two modes. During stable operation, the alarm screen is the first place to look if something abnormal happens before searching interactively. During a startup, on the other hand, the screen is used as a kind of check list; removal of alarms from the screen implies getting closer to a working machine. In both cases, the integrity of the system is of great importance. An alarm system in which one does not have 100% trust is almost useless. First for SPS and later for LEP, it took years of operational experience to get a viable system which the operators could use with confidence.

## 4 HOW CAN OPERATIONS GET THE FUNCTIONALITY THAT THEY WANT ?

An accelerator is a collection of many different systems, and a common approach is to provide applications software on a system by system basis. Working to agreed standards as outlined in the previous section, it should be reasonably straightforward for operations to specify what is needed sufficiently well for personnel outside operations to produce these applications. Many accelerator control systems are produced in this way, in some cases with a high degree of success [2]. Of course, all the underlying facilities of networks, communication protocols and so on have to be present, but this is taken for granted here.

However, if the operations team wants a coherent and integrated suite of applications, specification of the required functionality becomes much harder and can only be achieved if the whole system is conceived as a whole. This is an important step to take, and it is in this domain where SL division at CERN has had good experience of teams involving operations people at all stages of the development process, first for the SPS and later for LEP [3]. A brief history of this experience is described in some detail for the SPS, with additional remarks about the LEP software development where differences arose.

I should point out that in SL division, operations means people who work regularly on the machines but who do have time available, particularly during long machine shutdowns, for other things such as software development. Furthermore some of the operations personnel are engineers or physicists with several years of hands-on experience of running the machine(s).

### *A. SPS*

The SPS, commissioned in 1976, ran purely as a proton machine for a fixed target physics program until the early 80s. Then followed several years where about half of each year was devoted to proton-antiproton collider operation, mostly at a stable beam energy but also in 1985 as a pulsed collider allowing higher energy collisions to be achieved. This evolution meant many modifications, not least to the application programs, which underwent considerable and somewhat piecemeal development to accommodate the different modes of operation. This resulted in software that did the job, but that became difficult to maintain and almost impossible to modify. Consequently in 1985, when considerations began for further important changes to allow SPS to be used as the injector for LEP, it soon became clear that a major software rewrite was inevitable.

A small team of operations group physicists with a history of software development were mandated to look into the problem. The overall aim of the project was to provide, for early 1988, a software system for the operation of SPS with cycles of electron and positron bunches for LEP interleaved with proton or ion acceleration cycles for the ongoing SPS physics program. This would involve a move to a completely new platform on which to run the new software. The mandate also called for the establishment of clear guidelines for production of any new applications software for SPS and at a later date for LEP. With such a major and wide-ranging project, it was decided to look outside the accelerator sector for ideas. Through the seventies and early eighties the software industry had seen the increasing imposition of discipline on development activities, resulting in turn in the techniques of structured programming, structured design and structured analysis. These techniques had been introduced not on theoretical grounds, but were based on the desirable aspects found in a wide range of large, successful projects. Furthermore there were already groups at CERN who were beginning to use these techniques, and it was decided to follow the recommendations coming from these sources [4].

The team began by performing a detailed structured analysis of the various applications existing at the time to drive the SPS, using dataflow diagrams to describe what the software did. This allowed a better understanding of the details of the running system and brought out clear areas that were in common to all accelerator systems. Requirements for the new system were then collected, and used together with the analysis of the old to produce a model of the desired software. This model, again expressed in terms of dataflow diagrams and now with a full supporting data dictionary, represented a detailed description of the functionality that the new software should have, and formed the cornerstone of the project from then on. It was not easy to produce, taking more than a year for a core team of three people with significant contributions from many more. It must be said, however, that part of this time was spent getting organised and developing expertise in a new field.

It was realised that one can only go so far in way of developing the functionality of the system before the structure of the underlying data has to be understood. Data modeling therefore was a distinct phase to be completed before any further progress could be made. This resulted in a fully normalised description of the data needed to run the SPS in multicycling mode, and largely defined the design of the on-line database to be implemented to drive the software.

Armed with a comprehensive model of the required functionality and underlying data, design could start. Since the problem was well understood and thoroughly described, it was straightforward to divide the system into several parts. An important aspect of the way this partition was made was that personnel from the analysis team undertook the responsibility for the design and eventual coding of the different subsystems. The team all agreed to continue following a rather formal approach, using structured design techniques to develop structure charts of the various parts of the system before producing any software modules. This meant that the project, running since mid 1985, had still to deliver a single line of code by late 1987, a detail that was not lost on the project management at the time !

The importance of having the same people involved throughout cannot be overstressed. It avoided potentially disastrous pitfalls in the handover between analysis and design, and it instilled in each individual a huge sense of purpose that the whole product, from ideas through to implementation, should be delivered on time. In this respect the motivation of the different members of the team was very high during the difficult period of implementation according to schedule.

Implementation was made incrementally, with the core system in place for the LEP octant injection test of summer 1988. Definition of the operator interface was among the last and quickest things to be done, and was achieved using initial specifications from operations, rapid prototyping and iterative development. The result was, and still is, an operator interface very much appreciated by all users [5]. All major parts of the system were in place for LEP commissioning in mid 1989, with more functionality added over the following years.

After several years, the suite of software discussed continues to form the heart of SPS applications and is still generally regarded by operations as excellent. Nearly all personnel involved in the original development are no longer at CERN, and maintenance and further development is now carried out from the controls group.

### *B. LEP*

After the first year of running LEP, it was clear to several people in operations that the application software used for commissioning would need considerable improvement for efficient long-term exploitation of the machine [6]. Following the example of the SPS project, an initiative was launched in 1990 to attack the problem. A small team was assembled, again with a nucleus of operations group physicists having a history of software development (3 of them) but in this case with early involvement from controls personnel (a further 3). Functional decomposition of the problem was performed along similar lines to the SPS analysis, until once again it was clear that the data structure needed to be clearly defined. In this case entity relationship modeling techniques were used to organise the data, leading to a database design that was well suited to a commercial relational database system and the choice of ORACLE for implementation [7].

In a similar way to the SPS the system was then divided between several people on the team, but in this case the development from there on was not required to follow any (semi-)formal methods. Some people chose to use structured design to develop their ideas, others did not. More important was again the clear definition of responsibility for the different parts of the system, and the motivation and determination to deliver on time.

Initial implementation of the LEP software for the 1992 startup was more sweeping, reflecting a more complete suite of applications software, than in the case of the SPS. The operator interface drew heavily on both the ideas and the existing packages from the SPS, with all new developments made to the same standards, resulting in a highly uniform product across two very different machines. Developments since 1992 have mostly been to accommodate different modes of operation of the machine involving either new accelerator systems or existing ones driven in a new way. The main exception was the introduction in 1994 of a comprehensive beam measurement and fixed display system, now considered essential for routine operation.

### *C. Some comments*

The two development projects described had somewhat different origins. The first was a necessity in that something major had to be done in order to use SPS as LEP injector, while the second arose largely out of operations' discontent over the tools that were (not) available to do their job. However, in both cases it was felt that an approach

covering the whole accelerator was needed to produce applications software that was more coherent than in the past. Having decided this, it was clear that operations were best placed to define what the software should do. One may question whether it is necessary to be so ambitious, but in both cases the approach turned out to be justified in that it has resulted in an integrated suite of data-driven application software. As well as the undeniable advantage of having such an integrated system, there have been additional benefits such as that on both machines a new accelerator system can often be introduced without writing any code at all.

It is difficult to see how such accelerator-wide facilities could be specified without significant involvement of the people who drive the machine, but should their involvement stop there ? Ideally for operations the answer is yes, but in the two cases discussed the output of the analysis phase, while comprehensive, was very difficult to understand. This was particularly true in the SPS case, which perhaps represents a failing since one of the claims of structured decomposition is to control the complexity of a large system. In any case, to hand a specification of this kind over to a design team to develop and implement would introduce considerable overheads in the development process. Whether done formally or not, there are several iterations required around the analysis/design loop. In fact, this is the most critical phase of any project whose duration runs to several man-years. If the specifications and the implications thereof are not fully understood by the design and implementation team, the delivered product will never be up to standard. Having the same people involved throughout will at the very least allow important savings at this stage, and at most could prove the salvation of the project.

An illustration of this particular problem is to be found in the original software development for LEP, where one group of people produced specifications and handed them over to a second group for design and implementation. While the specifications contained all the ideas of the commissioning team, something was lost in the handover and a design of the *specified* software was never done. This resulted in a minimal system used to commission the accelerator, all of which was subsequently rewritten (the LEP development mentioned above).

### *D. Use of formal methods*

The formal methods adopted in the SPS and LEP projects were always seen as a means of getting the job done, not because of any imposed way of working. In fact in both projects, once the software was in place the analysis and design documents have not been kept up to date with the code. This of course means that these documents no longer describe what the software does and how, and maintenance is still a potential problem if personnel leave.

Having said this, it is difficult to see how either of the two projects described could have succeeded without the use of some formal technique or other. Structured analysis and design may not be the best method; it certainly has several weaknesses and to follow it rigorously could cause a project to lose momentum and stagnate.

A method, like any other tool, has to be understood and adapted to suit the needs of the situation. There is sometimes a tendency to insist on strictly following industry standards when it is not necessary to do so, and this can do more harm than good as the following example shows. An equipment specialist produced a rather simple specification for a piece of software to be developed by an application programmer. After some months this had been developed into a heavy formal document that was handed back to the equipment specialist for approval before development could begin in earnest. This required a significant investment by the end user that he was simply not prepared to give since he did not see the point. This project faltered to a halt because of this and had to be restarted (from the original simple specification).

## 5 CONCLUSIONS

The operations group should be regarded as the most important client of an accelerator control system simply because over the lifetime of the facility they are by far the main users of the system. However, at the start of routine operations the applications available are usually inherited from equipment and machine specialists who used them during the construction and commissioning phases. These applications are generally not sufficient for operations and a new software project is then needed. Whether or not this can be avoided is not clear. It may be very difficult to say in advance what is needed for routine operation, although there is plenty of experience available from running machines for guidance. What *can* be done is to recognize the existence of the problem and ensure that sufficient resources are available, be it sooner or later, to provide the required software.

There are many basic features that any application software should have which would make the operators' work much easier and therefore more efficient. Many of these features are often overlooked during the software development process.

Small development projects, say those measured in a few man-months, should be rather easy to handle. There is no need to consider formal methods if the user and the developer can agree on a specification by other means. A clear set of guidelines and a simple specification should in many cases suffice. In fact, introducing formal methods in such a case may achieve nothing more than slowing the development down, even to a standstill.

For large projects, say those measured in man-years, some kind of formal technique should certainly be considered. However, if you *are* planning to use formal techniques you had better learn how to use them. Prototype something from beginning to end before embarking on a big project. It is very easy to get stuck and produce nothing. Never try to impose formal techniques on unwilling members of the team.

It is important for the project management to realise that there are certain distinct phases in any large development. Functional decomposition and data modeling are best done on a project-wide basis, and should be essentially complete before partitioning is considered. Subsequent design, coding and implementation techniques are best left to the people or teams concerned, unless the eventual maintenance of the software is considered a serious problem. If for any reason this is an area of concern, it may well be necessary to require documentation in some serious form, which would strengthen the case for using formal techniques further into the project.

For projects covering many aspects of the accelerator, the software development team must include operations people at all stages of the development process. They are certainly needed to produce the detailed specification of the software, and continued involvement allows many corners to be cut in the later phases of the development. In both of the projects discussed in this paper operations people produced most of the code, which is not ideal for many reasons but bypassed the problem of handing off specifications or designs to others. A better model of how to staff such a development would see a sharp decline in operations' involvement through the design phase, with a corresponding increase in effort from software engineers from controls (see Figure 1). This would open up the way for proper management of the finished code, allowing version control and maintenance in general to be handled fully by controls.

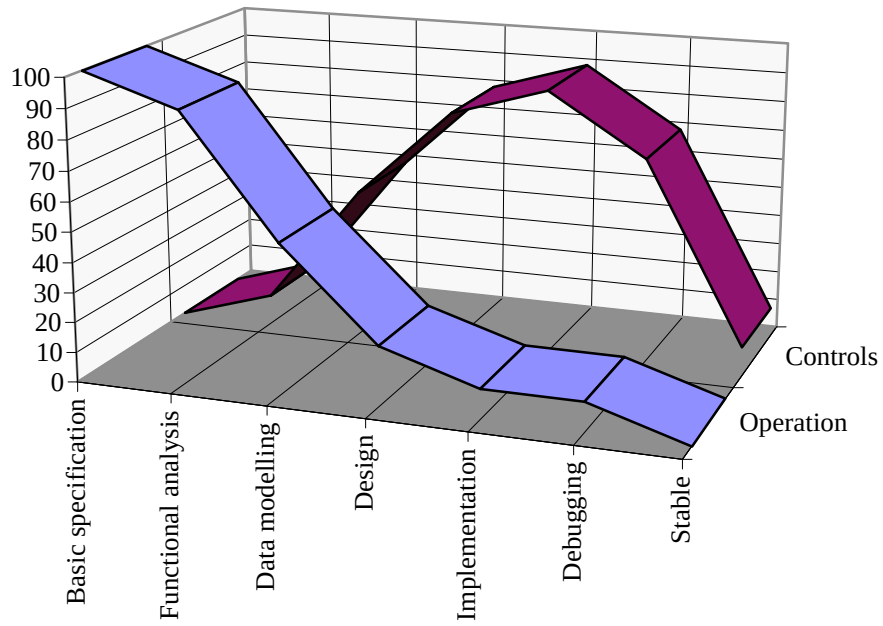


**Figure 1: Ideal evolution of operational and controls involvement through the various phases of a large applications software project**

Operations people are probably best placed to perform the overall coordination of large application software projects, because they have an accelerator-wide view of the problems, because they know what they want and because they have a vested interest in getting the end product to use. Alternatively this role could be played by a software engineer from the controls group, but to be effective he or she would have to develop a sound understanding of what controlling an accelerator is all about.

Finally a word of warning; operations cannot and should not try to do everything. In many laboratories the operations staff have little or no time for software development, and even in the best cases their availability is limited and sporadic. Significant and sustained support is needed by full time software professionals. With the right balance and the right spirit of collaboration, impressive results can be achieved. Finding this balance continues to be the elusive goal for the manager of any large software project in accelerator control.

## 6. REFERENCES


[1] E. McCrory, these proceedings
[2] M. Stanek, these proceedings
[3] M. Lamont, these proceedings
[4] I.T. Wilkie et al, Proc. ICALEPCS, Villars 1987
[5] A. Ogle et al, Proc. ICALEPCS, Vancouver 1989
[6] R. Bailey, Proc. ICALEPCS, Tsukuba 1991
[7] R. Bailey et al, ICALEPCS, Berlin 1993